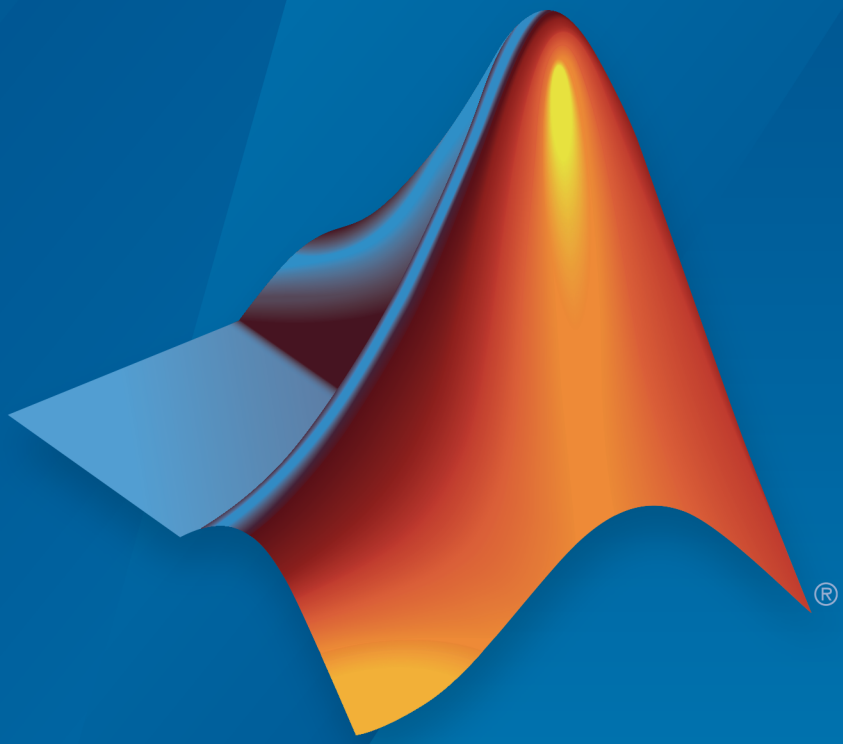
Simulink® Code Inspector™ Release Notes



MATLAB® & SIMULINK®

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

Simulink® Code Inspector™ Release Notes

© COPYRIGHT 2011–2015 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Contents

R2015a

Support for MATLAB Function block 1-2

Optional checksum for referenced models compiled as top . 1-2

R2014b

Support for structured parameter objects 2-2

Support for Bias and Unary Minus blocks 2-2

Enhanced support for Enable Port and Trigger Port blocks 2-2

R2014a

Support for data type replacement 3-2

Support for Dot Product and Polynomial blocks 3-2

Support for graphical functions in Stateflow charts 3-2

R2013b

Support for Stateflow objects and classic action language . . 4-2
Stateflow compatibility checks 4-2
Stateflow code inspection status in report 4-2

Support for Assignment and Bus Assignment blocks 4-3

Enhanced support for Simulink blocks and model options, including MinMax and Product block input options 4-3
Simulink blocks 4-3
Model Referencing options for testpointed root Outport blocks 4-4
Diagnostics: Data Validity options for precision loss 4-4
Diagnostics: Model Referencing options for invalid root Inport/Outport blocks 4-4
Code Generation: `AdvancedOptControl` setting now FATAL incompatibility 4-4

R2013a

Support for Stateflow data and transitions 5-2
Stateflow data 5-2
Stateflow transition code inspection status in report 5-2

Support for subsystems and models with active design variants 5-2

Support for models that generate nonreusable or nonreentrant code 5-2

Support for Width, Dynamic Dead Zone, and Dynamic Saturation blocks 5-2

Additional compatibility checks 5-3

R2012b

Support for Simulink subsystems: If Action Subsystem, Function-Call Subsystem, and Triggered Subsystem blocks 6-2

Support for Simulink blocks: Discrete Time Integrator, Bitwise Operator, Shift Arithmetic, Sqrt, and others 6-2

Support for 2D matrix data type and tunable parameter . . . 6-2

Support for C99 and code replacement math library 6-2

Enhanced code inspection 6-3

DO-178C qualification support (requires DO Qualification Kit) 6-3

Additional compatibility checks 6-3

R2012a

Support for Additional Simulink Blocks 7-2

Enhanced Sum, MinMax, and Logical Operator Block Support 7-2

Support for Enumerated Types 7-2

Support for Nonencapsulated C++ Code 7-2

Traceability Matrix Generation 7-2

DO-178 Qualification Support (Requires DO Qualification Kit) 7-3

Enhanced Support for Code Optimizations 7-3

Introducing Simulink Code Inspector 8-2

Features 8-2

Introductory Example 8-2

R2015a

Version: 2.3

New Features

Bug Fixes

Support for MATLAB Function block

Code inspection is supported for MATLAB Function blocks and a subset of the MATLAB® language in MATLAB Function block scripts. In R2015a, the following MATLAB language elements are supported:

- MATLAB language expressions for scalar data.
- Subscript operators for array data.
- MATLAB language statement.

For more information on the MATLAB language elements that are supported, see "MATLAB Function Block Support".

Blocks supported for code inspection are available in the block library `slcilib`. To open this library, in the Command Window, enter slcilib.

Optional checksum for referenced models compiled as top

Previously, when a model was inspected as a referenced model, Simulink® Code Inspector™ computed the checksum for the referenced model compiled as a referenced model. The checksum was then displayed in the code inspection report. In R2015a, Simulink Code Inspector can compute and report a checksum for the referenced model compiled as a top model in addition to the referenced model checksum. This option provides you with a checksum consistent with checksums computed by other MathWorks products that compile models in the same context.

In the Simulink Code Inspector dialog box, you can choose to include the top model checksum. Select the **Include additional checksum for referenced model(s) compiled as top** check box when either of the following is true:

- You do not select the **This is the top of the model hierarchy** check box.
- You select the **Inspect all referenced models** check box.

The **Include additional checksum for referenced model(s) compiled as top** check box is cleared by default.

To select this option from the command line, use the new API `slci.Configuration.setIncludeTopModelChecksumForRefModels`.

R2014b

Version: 2.2

New Features

Bug Fixes

Support for structured parameter objects

Code inspection is supported for structured parameter objects. Previously, tunable workspace variables with structured data types were not compatible with code inspection. For more information, see Model-Wide Attribute Constraints.

Support for Bias and Unary Minus blocks

Code inspection is supported for the following Simulink blocks:

- Bias
- Unary Minus

Blocks supported for code inspection are available in the block library `slcilib`. To open this library, in the Command Window, enter slcilib.

Enhanced support for Enable Port and Trigger Port blocks

Code inspection now supports Enable Port and Trigger Port blocks that output the enabling signal. Previously, selecting Enable Port and Trigger port block parameter **Show output port** (`ShowOutputPort`) was a nonfatal incompatibility with code inspection. For more information, see Enable Port and Trigger Port.

R2014a

Version: 2.1

New Features

Bug Fixes

Support for data type replacement

Previously, models with data type replacement tables were not compatible with code inspection. In R2014a, code inspection is supported for models that replace data type names in the generated code using the **Data type names** table. The **Data type names** table is on the **Code Generation > Data Type Replacement** pane in the Configuration Parameters dialog box. The compatibility check **Check code generation settings > Verify 'Replace data type names in the generated code' setting** is removed.

The **Replacement Type Usage** section of the code inspection report provides the data type replacement verification results for each Simulink built-in data type. The **Replacement Type Usage** section flags code generation types in the generated code that are not replaced by the replacement type name. This flagging includes types used when declaring symbols, cast, and size of expressions. However, the section does not flag code generation types in shared utility files such as `rtwtypes.h`.

Support for Dot Product and Polynomial blocks

Code inspection is supported for the following Simulink blocks:

- Dot Product
- Polynomial

Blocks supported for code inspection are available in the block library `slcilib`. To open this library, in the Command Window, enter slcilib.

Support for graphical functions in Stateflow charts

Code inspection is supported for graphical functions in Stateflow® charts.

For constraints that apply, see Stateflow Graphical Functions.

R2013b

Version: 2.0

New Features

Bug Fixes

Support for Stateflow objects and classic action language

Simulink Code Inspector supports the following Stateflow objects for code inspection:

- Boxes
- Charts
- Data and Events
- Junctions
- Non-Atomic Subcharts
- Simulink functions
- States
- Transitions

For constraints that apply, see Stateflow Object Constraints.

Stateflow compatibility checks

To verify that Stateflow objects in your model are compatible with code inspection, use these Stateflow compatibility checks:

- Check for Stateflow machine data
- Check for Stateflow machine events
- Check usage of Stateflow blocks
- Check usage of Stateflow charts
- Check usage of Stateflow transitions
- Check usage of Stateflow junctions
- Check usage of Stateflow data
- Check usage of Stateflow events
- Check usage of Stateflow states

Stateflow code inspection status in report

Simulink Code Inspector reports the traceability and verification status of Stateflow states, charts, and transitions. Previously, code inspection reported only the status of Stateflow charts and transitions.

Support for Assignment and Bus Assignment blocks

Code inspection is supported for the following Simulink blocks:

- Assignment
- Bus Assignment

Blocks supported for code inspection are available in the block library `slcilib`. To open this library, in the Command Window, enter slcilib.

Enhanced support for Simulink blocks and model options, including MinMax and Product block input options

Simulink blocks

The following table indicates code inspection enhancements to Simulink blocks.

Simulink block	Code inspection support enhancement
1-D Lookup Table, 2-D Lookup Table, n-D Lookup Table (1 or 2-D)	Out-of-range input in generated code.
Multiport Switch	2 or more inputs. Previously, code inspection was supported only for blocks with 3 or more inputs.
Product	• 2 or more inputs for `element-wise` multiplication. Previously, code inspection was supported only for blocks with 2 inputs. • Integer matrix multiplication. Previously, only single and double data type inputs were supported.
Selector	When **Index Option** (`IndexOptionArray`) is set to `Index vector (port)`, code inspection is now supported.
MinMax, Sum, Add, Subtract	1 or more inputs. Previously, code inspection was supported only for blocks with 2 or more inputs.

For information, see Supported Block Constraints. Blocks supported for code inspection are available in the block library `slcilib`. To open this library, in the Command Window, enter slcilib.

Model Referencing options for testpointed root Outport blocks

Code inspection now supports testpointed root Outport blocks in referenced models. Constraints on the Configuration Parameter **Model Referencing > Total number of instances allowed per top model** (`ModelReferenceNumInstancesAllowed`) and compatibility check **Check for root Outport blocks being testpointed** are removed.

Diagnostics: Data Validity options for precision loss

Code inspection now supports parameters with precision loss during simulation. Previously, Simulink Code Inspector recommended that you set Configuration Parameter **Diagnostics > Data Validity > Detect precision loss** (`ParameterPrecisionLossMsg`) to `error`. For other constraints on the **Diagnostics > Data Validity** Configuration Parameters, see Diagnostics: Data Validity.

Diagnostics: Model Referencing options for invalid root Inport/Outport blocks

Code inspection now supports the automatic insertion of hidden signal copy blocks at the model inports and outports. Previously, Simulink Code Inspector recommended that you set Configuration Parameter **Diagnostics > Model Referencing > Invalid root Inport/Outport block connection** (`ModelReferenceIOMsg`) to `error`. Setting **Invalid root Inport/Outport block connection** to `error` configured your model to generate an error if Simulink detected invalid internal connections to the root-level Inport or Outport blocks. If your model generated this error, Simulink Code Inspector recommended that you manually insert Signal Conversion blocks to avoid the error and maintain traceability.

Code Generation: `AdvancedOptControl` setting now FATAL incompatibility

For compatibility with code inspection, configuration parameter `AdvancedOptControl` must be set to `-SLCI`. Not setting `AdvancedOptControl` to `-SLCI` is now a FATAL incompatibility. Previously, this setting was a nonfatal incompatibility. For more information, see Fatal Incompatibilities.

R2013a

Version: 1.3

New Features

Bug Fixes

Support for Stateflow data and transitions

Stateflow data

Code inspection is now supported for Stateflow charts that use:

- Data stores
- Constants
- Parameters
- Local data
- Nonscalar data

Previously, only Stateflow charts with scalar inputs and outputs were supported for code inspection. For detailed support information, see Stateflow.

Stateflow transition code inspection status in report

Simulink Code Inspector reports the traceability and verification status of Stateflow charts and transitions. Previously, code inspection reported only the status at the chart level.

For more information, see Code Inspection Reports.

Support for subsystems and models with active design variants

Code inspection now supports models and subsystem blocks with active design variants. Previously, if your model or subsystem block had design variants, the model or subsystem was not compatible with code inspection. For detailed support information, see the code inspection Model and Subsystems constraints.

Support for models that generate nonreusable or nonreentrant code

Code inspection now supports models that generate nonreusable and nonreentrant code. Previously, compatibility with code inspection, models had to generate reusable and reentrant code.

Support for Width, Dynamic Dead Zone, and Dynamic Saturation blocks

Code inspection is now supported for the following Simulink blocks:

- Width
- Dynamic Dead Zone
- Dynamic Saturation

The Dynamic Dead Zone and Dynamic Saturation blocks are mask blocks. For a complete list of mask blocks supported for code inspection, see Supported Mask Blocks. Blocks that are supported for code inspection are available in the block library `slcilib`. You can open this library by entering slcilib in the MATLAB Command Window.

Additional compatibility checks

For further verification that your model is compatible with Simulink Code Inspector, there are additional compatibility checks:

- Check for Terminator blocks connected to Model Reference block outports
- Check for root Outport blocks being testpointed

R2012b

Version: 1.2

New Features

Bug Fixes

Support for Simulink subsystems: If Action Subsystem, Function-Call Subsystem, and Triggered Subsystem blocks

Code inspection is now supported for the following Simulink subsystem blocks:

- If Action Subsystem
- Function-Call Subsystem
- Triggered Subsystem

For detailed block support information, see Block Constraints.

Support for Simulink blocks: Discrete Time Integrator, Bitwise Operator, Shift Arithmetic, Sqrt, and others

Code inspection is now supported for the following Simulink blocks.

Action Port	Bitwise Operator	Data Type Propagation	Discrete-Time Integrator
Function-Call Generator	If	Merge	Probe
Reshape	Shift Arithmetic	Sign	Signal Specification
Sqrt	Switch Case	Trigger	Vector Concatenate

For detailed block support information, see Block Constraints.

Support for 2D matrix data type and tunable parameter

Code inspection is enhanced for blocks with 2D matrix data types.

Code inspection no longer requires you to select **Optimization > Signals and Parameters > Inline parameters** on the Configuration Parameters dialog box. In addition, you can now use symbolic names or inlined numerical values for tunable model parameters in generated code. Previously, tunable parameters were constrained to inlined numerical values.

Support for C99 and code replacement math library

You can now inspect code that uses C99 libraries and Supported Functions and Operations in Code Replacement Libraries.

Enhanced code inspection

Code inspection is enhanced for:

- Trigonometric Function blocks. Code inspection now supports the `sincos` function.
- Product and Gain blocks. Code inspection now supports matrix multiplication.
- Blocks with function-calls at root of model.
- Switch Case Action Subsystem.

For detailed block support information, see Block Constraints.

DO-178C qualification support (requires DO Qualification Kit)

The DO Qualification Kit product now provides documents, templates, test cases, and test procedures that you can use to qualify the Simulink Code Inspector tool for DO-178C certification.

Additional compatibility checks

There are additional checks to verify that your model is compatible with Simulink Code Inspector:

- Check storage class for workspace variables
- Check for usage of synthesized local data stores
- Check loop unrolling threshold setting
- Check destinations of If and Switchcase blocks

For detailed information on using the checks, see Model Compatibility.

R2012a

Version: 1.1

New Features

Bug Fixes

Support for Additional Simulink Blocks

Code inspection is now supported for the following Simulink blocks:

- Atomic Subsystem (inlined)
- Enabled Subsystem (inlined)
- Enable Port
- 1-D Lookup Table
- 2-D Lookup Table
- n-D Lookup Table (1 or 2-D only)
- Rounding Function
- Ground

For detailed block support information, see Block Constraints.

Enhanced Sum, MinMax, and Logical Operator Block Support

Code inspection now supports greater than two inputs for Sum, MinMax, and Logical Operator blocks.

Support for Enumerated Types

Code inspection now supports enumerated types used in models.

Support for Nonencapsulated C++ Code

Code inspection now supports models for which the selected target language is `C++`, as well as `C`. The target language `C++ (Encapsulated)` remains unsupported.

Traceability Matrix Generation

On Windows® systems, R2012a allows you to generate a *traceability matrix* for your model. For a given model, a generated traceability matrix provides information about traceability of model objects between the model and generated code. The traceability matrix is a Microsoft® Excel® file that contains **Model Information**, **Code Interface**, **Code Files**, and **Report** worksheets.

After generating code and inspecting a model, you can generate a traceability matrix using the `slci.ExportTraceReport` function from the MATLAB Command Window. For example:

```
>> slci.ExportTraceReport('slcidemo_roll','slcidemo_roll_tracereport')
```

For more information, see Traceability Matrices.

DO-178 Qualification Support (Requires DO Qualification Kit)

The DO Qualification Kit product now provides documents, templates, test cases, and test procedures that you can use to qualify the Simulink Code Inspector tool for DO-178B certification.

Enhanced Support for Code Optimizations

Code inspection now supports any setting for the following code optimizations, which are located on the **Optimization** and **Optimization > Signals and Parameters** panes of the Configuration Parameters dialog box. Previously, their values were constrained to either `on` or `off`.

- **Remove root level I/O zero initialization** (`ZeroExternalMemoryAtStartup`)
- **Remove internal data zero initialization** (`ZeroInternalMemoryAtStartup`)
- **Use memset to initialize floats and doubles to 0.0** (`InitFltsAndDblsToZero`)
- **Use memcpy for vector assignment** (`EnableMemcpy`)

R2011b

Version: 1.0

New Features

Introducing Simulink Code Inspector

Simulink Code Inspector automatically compares generated code with its source model to satisfy code-review objectives in DO-178B and other high-integrity standards. The Code Inspector systematically examines blocks, parameters, and settings in a model to determine whether they are structurally equivalent to operations, operators, and data in the generated code. Simulink Code Inspector provides detailed model-to-code and code-to-model traceability analysis. It generates structural equivalence and traceability reports that you can submit to certification authorities to satisfy DO-178 software coding verification objectives.

Features

Key features of Simulink Code Inspector Version 1.0 include:

- Structural equivalence analysis and reports
- Bidirectional traceability analysis and reports
- Compatibility checker to restrict model, block, and coder usage to operations typically used in high-integrity applications
- Tool independence from Simulink code generators

Use Simulink Code Inspector tooling to:

- Prepare for code inspection during model development.
- Run inspections on code generated from models and review reported results.
- Automatically generate code verification reports to support software certification.

Introductory Example

The Simulink Code Inspector product provides the following introductory example.

Example	Shows How You Can...
slcidemo_intro	Use MATLAB commands to: • Prepare a model hierarchy for code generation and code inspection. • Automatically generate code for the model hierarchy.

Example	Shows How You Can...
	• Verify the generated code independently of the code generation tool. • Purposely introduce an error into the generated code and inspect for failure.

Note: Inspect Generated Code for a Sample Model in the Simulink Code Inspector documentation provides an equivalent example using the Simulink Code Inspector dialog box to control the code inspection workflow.